Alessandro Mengoli

Costruire e orchestrare sistemi multi-agent con Microsoft Agent Framework

È IN PREVIEW

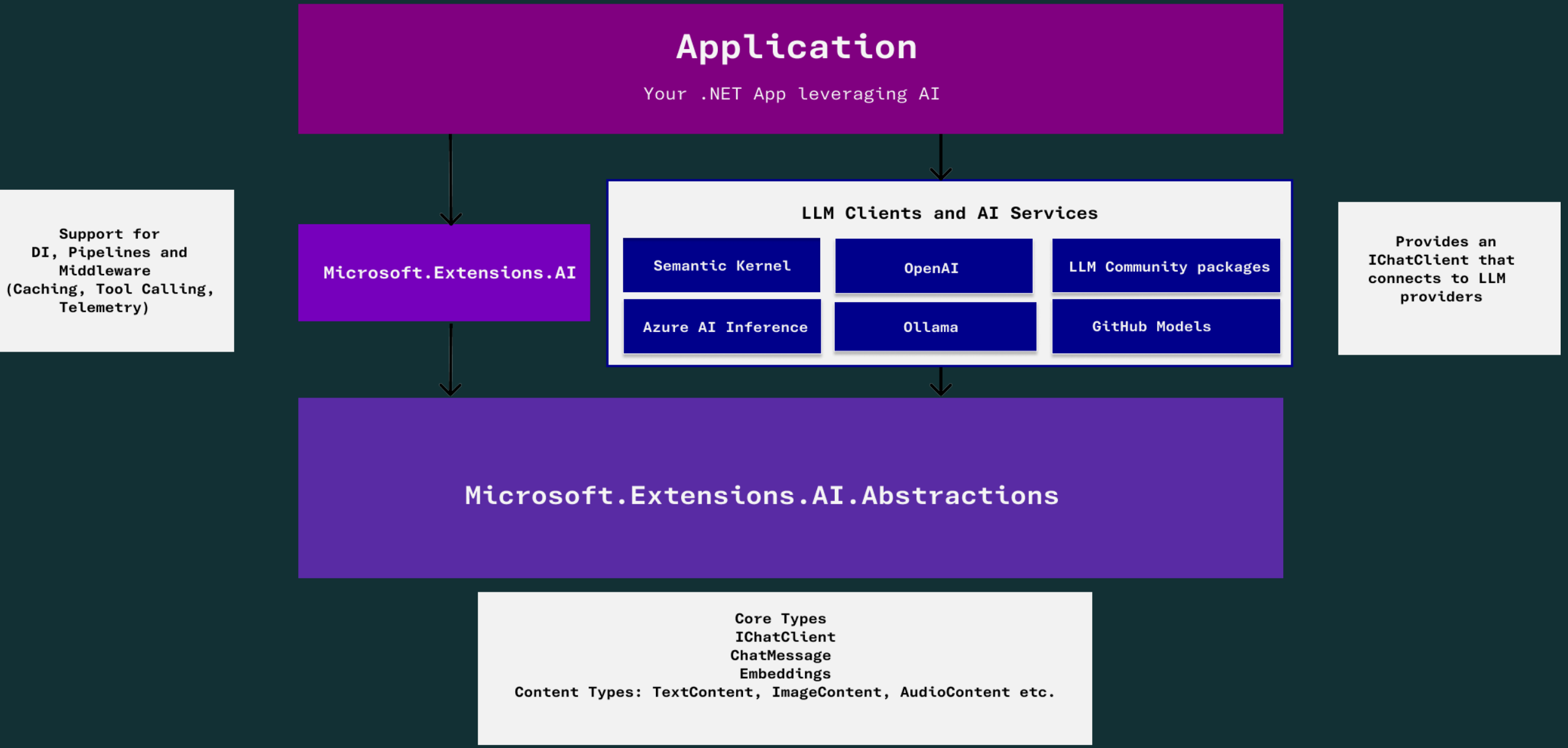
È IN PREVIEW

SEMANTIC KERNEL

AUTOGEN

Microsoft.Extensions.AI

"La libreria Microsoft.Extensions.AI forniscono un approccio unificato per rappresentare i componenti di AI generativa e consentono un'integrazione fluida e l'interoperabilità con vari servizi di AI."


Application
Your .NET App leveraging AI
Support for DI, Pipelines and Middleware (Caching, Tool Calling, Telemetry)
Microsoft.Extensions.AI
LLM Clients and AI Services
Semantic Kernel
OpenAI
LLM Community packages
Azure AI Inference
Ollama
GitHub Models
Provides an IChatClient that connects to LLM providers
Microsoft.Extensions.AI.Abstractions
Core Types
IChatClient
ChatMessage
Embeddings
Content Types: TextContent, ImageContent, AudioContent etc.

Interfaces

IChatClient: defines a client abstraction responsible for interacting with AI services that provide chat capabilities. It includes methods for sending and receiving messages with multi-modal content

IEmbeddingGenerator: represents a generic generator of embeddings

IImageGenerator (experimental): represents a generator for creating images from text prompts or other input

```
using Microsoft.Extensions.AI;
using OllamaSharp;

IChatClient client = new OllamaApiClient( new Uri("http://localhost:11434/"), "phi3:mini");

Console.WriteLine(await client.GetResponseAsync("What is AI?"));
```

Livelli di astrazione

IChatClient

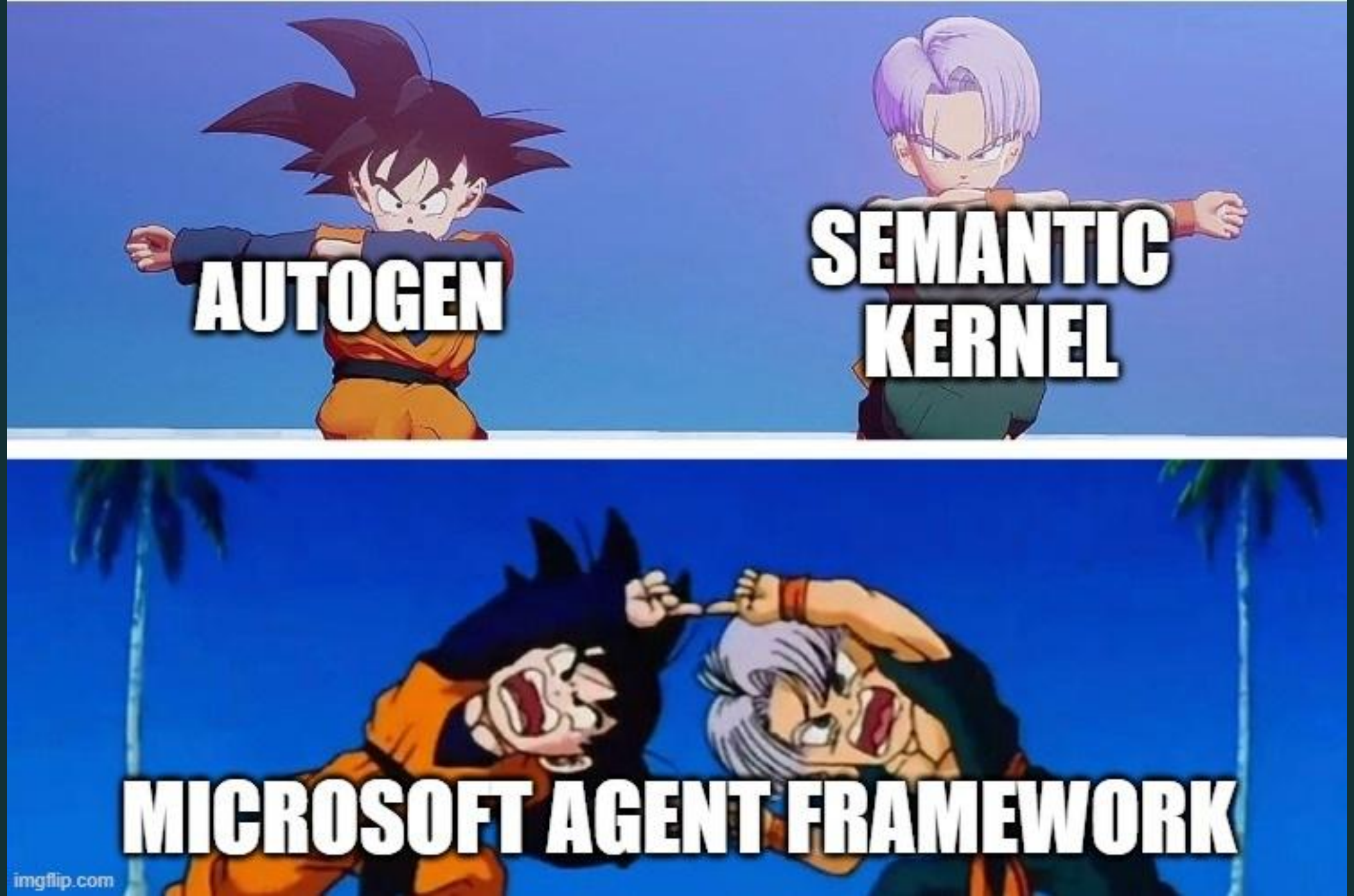
Microsoft.Extensions.AI

Quando usare IChatClient?

Quando basta una chiamata

- Hai bisogno di una singola interazione con un LLM
- Non ti serve gestione dello stato/conversazione
- Vuoi massima flessibilità e controllo basso livello
- Stai costruendo le tue astrazioni sopra

Se mi servisse qualcosa in più?


AUTOGEN
SEMANTIC KERNEL
MICROSOFT AGENT FRAMEWORK
imgflip.com

Microsoft Agent Framework

- Framework open-source per sviluppare agenti AI e workflow multi-agente in .NET e Python
- Unisce le astrazioni semplici di AutoGen con le funzionalità enterprise di Semantic Kernel

Novità chiave:

- Controllo esplicito sui flussi di esecuzione multi-agente
- Sistema robusto di gestione dello stato per scenari long-running e human-in-the-loop

Microsoft's agent frameworks

Orchestrate agents with **AutoGen, Semantic Kernel, Foundry SDK and M365 Agents SDK**

AutoGen

State-of-the-art multi-agent research SDK

Semantic Kernel

OSS AI service and agent orchestrator

Foundry SDK

API for Foundry Agent Service

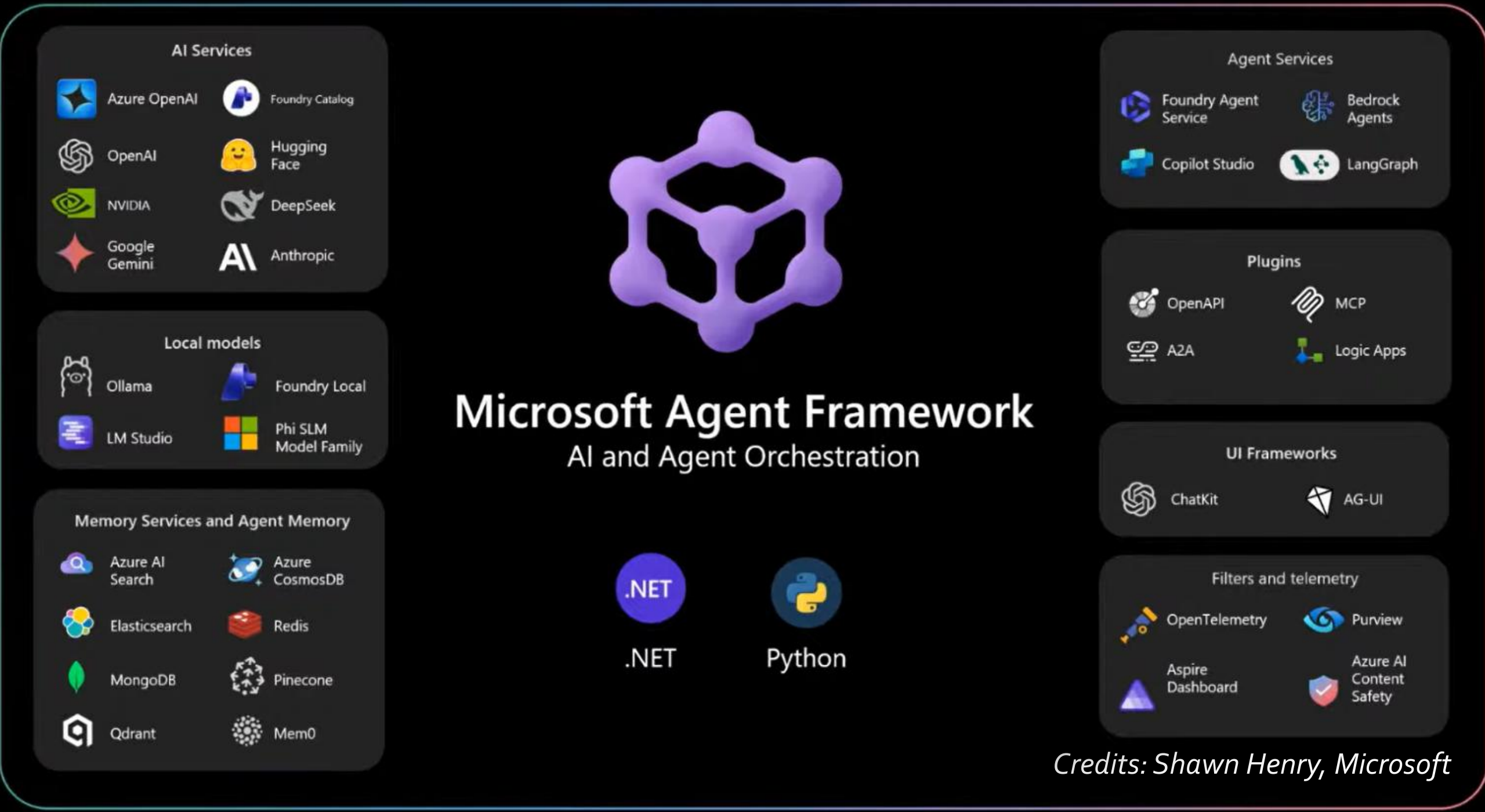
M365 Agent SDK

API for Foundry Agent Service

Unified Agent Development

Microsoft Agent Framework

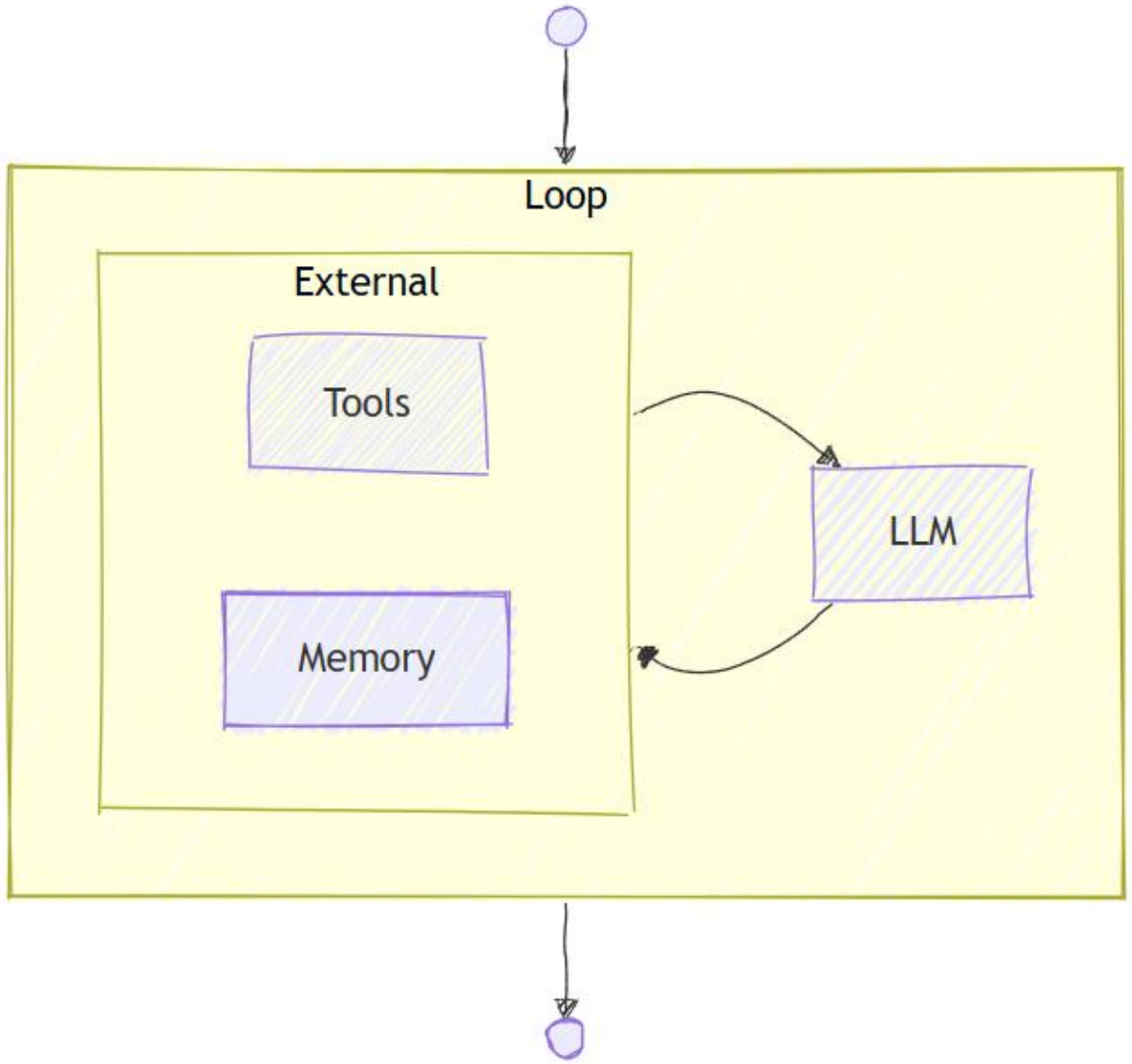
Credits: Shawn Henry, Microsoft



Credits: Shawn Henry, Microsoft

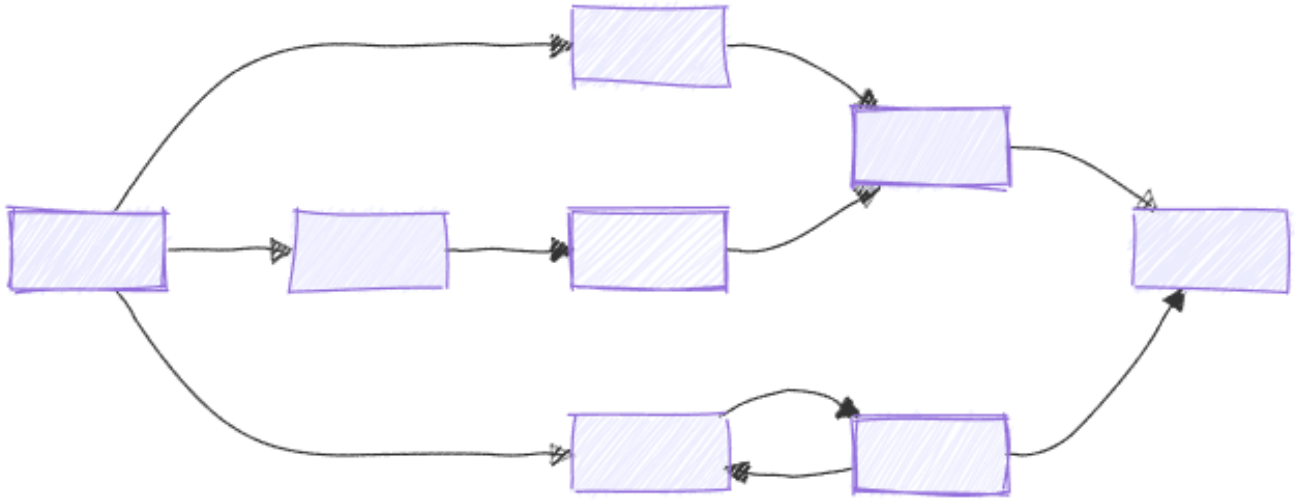
Due modalità

1) AIAgents: Agenti che usano LLM per elaborare input, chiamare tool e server MCP, generare risposte. Adatti per decisioni autonome e pianificazione ad-hoc e task non strutturati e imprevedibili

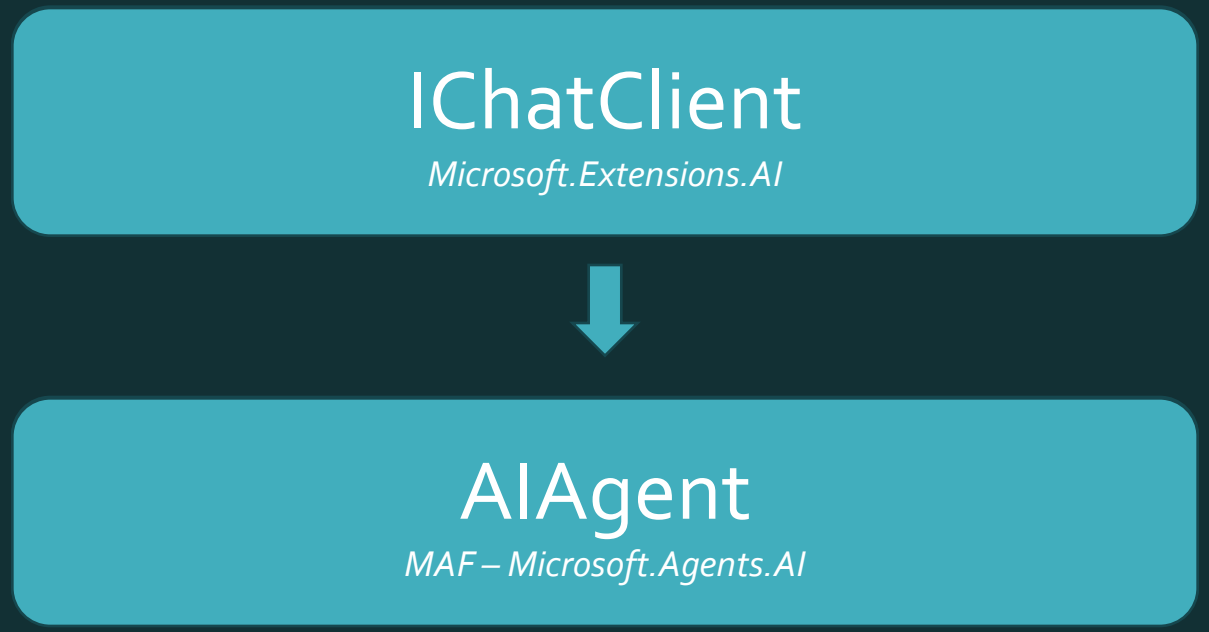


Due modalità

2) Workflows: Sequenze predefinite basate su grafi che connettono agenti, funzioni e sistemi esterni. Aggiunge orchestrazione, HITL, Checkpoint



Livelli di astrazione



DEMO

```
// L'agent wrappa l'IChatClient e aggiunge funzionalità
AIAgent agent = new ChatClientAgent(
    chatClient,
    instructions: "Sei un assistente per prenotazioni hotel. Sei cordiale e preciso.",
    name: "BookingAssistant",
    tools: [searchHotelsTool, bookRoomTool]
);

AgentThread thread = agent.GetNewThread();
var response1 = await agent.RunAsync("Cerco un hotel a Roma per 2 notti", thread);
var response2 = await agent.RunAsync("Preferisco 4 stelle", thread); // ricorda il contesto!
```

Feature	IChatClient	AIAgent
Chiamata singola	✅	✅
Multi-turn conversation	❌ (manuale)	✅ automatico
Function calling	❌ (manuale)	✅ built-in
Identità (nome, istruzioni)	❌	✅
Thread serialization	❌	✅
Middleware support	❌	✅

Ristrutturazione casa

Ristrutturazione casa

Possiamo chiedere qualche dritta iniziale per idee

Ristrutturazione casa

Possiamo chiedere qualche dritta iniziale per idee

Poi abbiamo due strade:

A) affidarci ad un ciappinaro che fa tutto

ciappianaro = una persona che fa lavoretti di riparazione, aggiustando un po' di tutto, spesso con una manualità pratica. Non è un professionista specializzato, ma uno che si arrangia con ingegno e buona volontà

Ristrutturazione casa

Possiamo chiedere qualche dritta iniziale per idee
Poi abbiamo due strade:

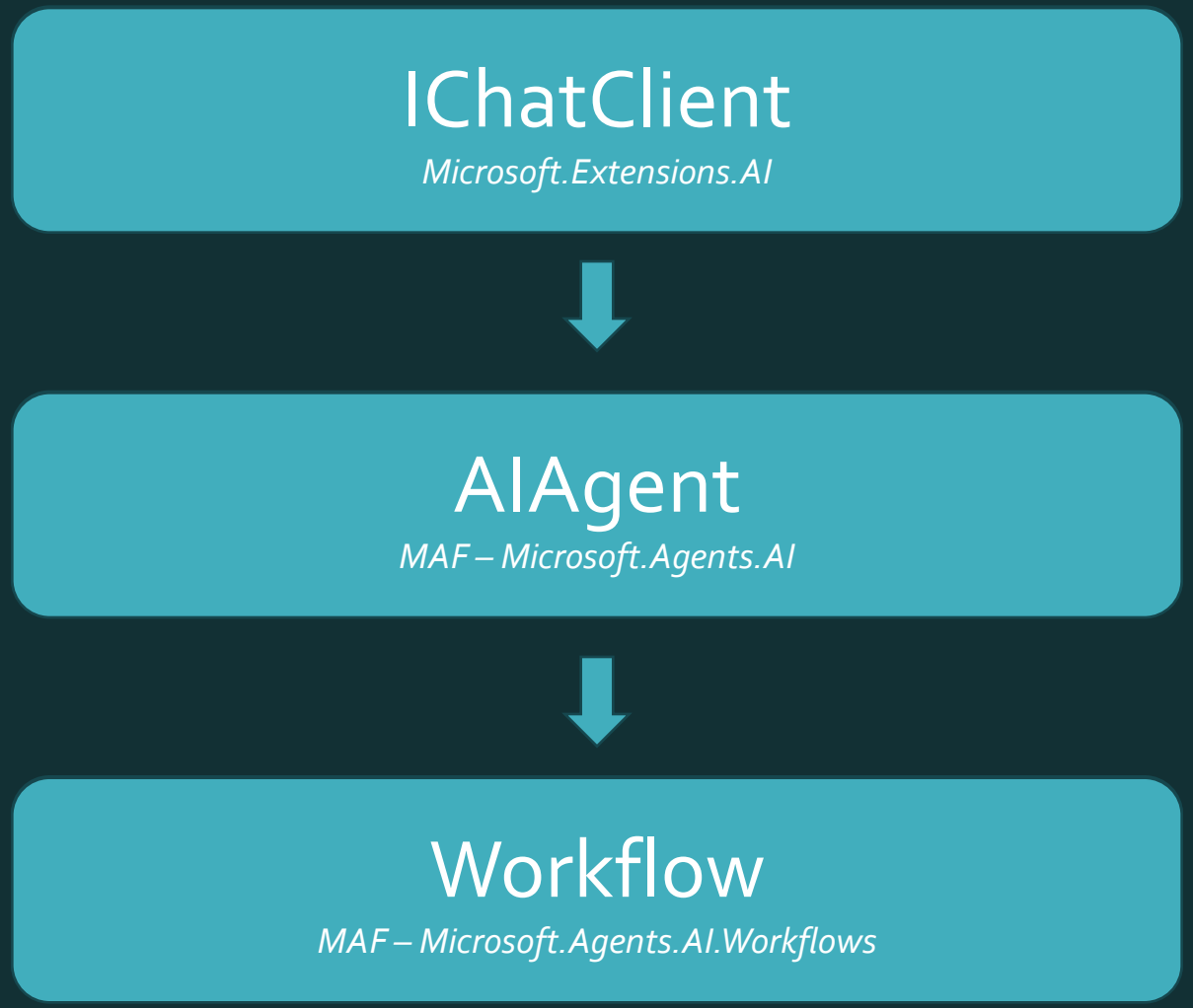
A) affidarci ad un ciappinaro che fa tutto

B) affidarci una serie di professionisti esperti:

- muratore
- geometra
- termotecnico
- idraulico

...

Livelli di astrazione



Due dritte/idee

Ciappinaro o singolo esperto

Team di professionisti

Workflows

- Modularità: componenti riutilizzabili
- Type Safety: validazione completa dei messaggi
- Flessibilità: routing condizionale, esecuzione parallela
- Checkpointing: salvataggio stato per processi long-running
- Orchestrazione multi-agente
- Human-in-the-loop
- Perfetti per: processi complessi multi-step con più agenti e decision point

Building blocks

- Executors: unità di elaborazione (agenti AI o logica custom). Ricevono input, eseguono task, producono output
- Edges: connessioni tra executors, definiscono il flusso dei messaggi. Possono includere condizioni
- Workflows: grafi diretti che orchestrano executors e edges
- Events: permetto di rendere osservabile il comportamento. Eventi di inizio, fine, errori e risposte degli executors

```
// Create agents
AIAgent spamDetectionAgent = GetSpamDetectionAgent(chatClient);
AIAgent emailAssistantAgent = GetEmailAssistantAgent(chatClient);

// Create executors
var spamDetectionExecutor = new SpamDetectionExecutor(spamDetectionAgent);
var emailAssistantExecutor = new EmailAssistantExecutor(emailAssistantAgent);
var sendEmailExecutor = new SendEmailExecutor();
var handleSpamExecutor = new HandleSpamExecutor();

// Build the workflow with conditional edges
var workflow = new WorkflowBuilder(spamDetectionExecutor)
    // Non-spam path: route to email assistant when IsSpam = false
    .AddEdge(spamDetectionExecutor, emailAssistantExecutor, condition: GetCondition(expectedResult: false))
    .AddEdge(emailAssistantExecutor, sendEmailExecutor)
    // Spam path: route to spam handler when IsSpam = true
    .AddEdge(spamDetectionExecutor, handleSpamExecutor, condition: GetCondition(expectedResult: true))
    .WithOutputFrom(handleSpamExecutor, sendEmailExecutor)
    .Build();
```

Orchestration

Pattern	Topologia	Chi decide il flusso?	Use Case
Sequential	Pipeline lineare	Predefinito (tu)	Task dipendenti, pipeline di processing
Concurrent	Fan-out/Fan-in	Predefinito (parallelo)	Analisi indipendenti, voting, ensemble
Group Chat	Star (hub centrale)	Manager (round-robin o custom)	Brainstorming, revisione iterativa
Magentic	Star con planner	Manager AI con planning dinamico	Problemi complessi e open-ended
Handoff	Mesh (peer-to-peer)	Gli agenti stessi	Customer support, escalation, routing

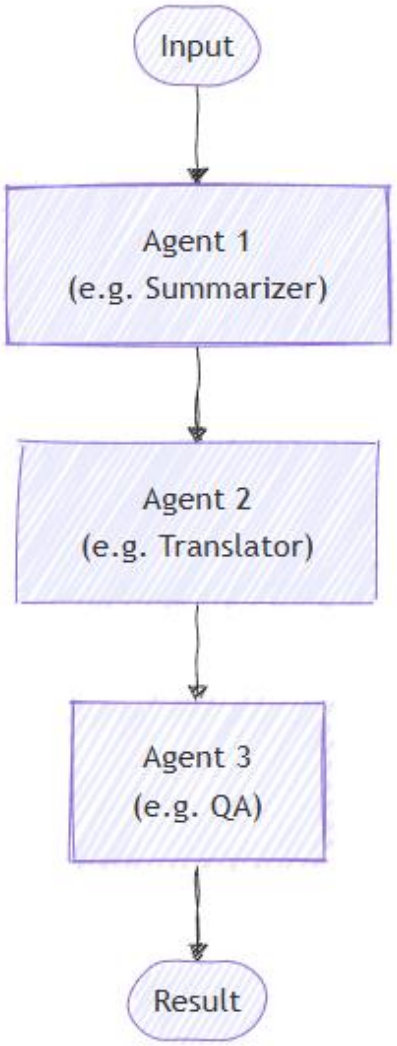
Sequential

Caratteristiche

- Ogni agente riceve l'intera storia della conversazione precedente
- L'ordine è strettamente rispettato
- Puoi mixare agenti AI con executor custom (logica non-AI)

Quando usarlo

- Task con dipendenze lineari (ogni step dipende dal precedente)
- Pipeline di trasformazione dati
- Workflow "draft→review→polish "
- Document processing multi-stage



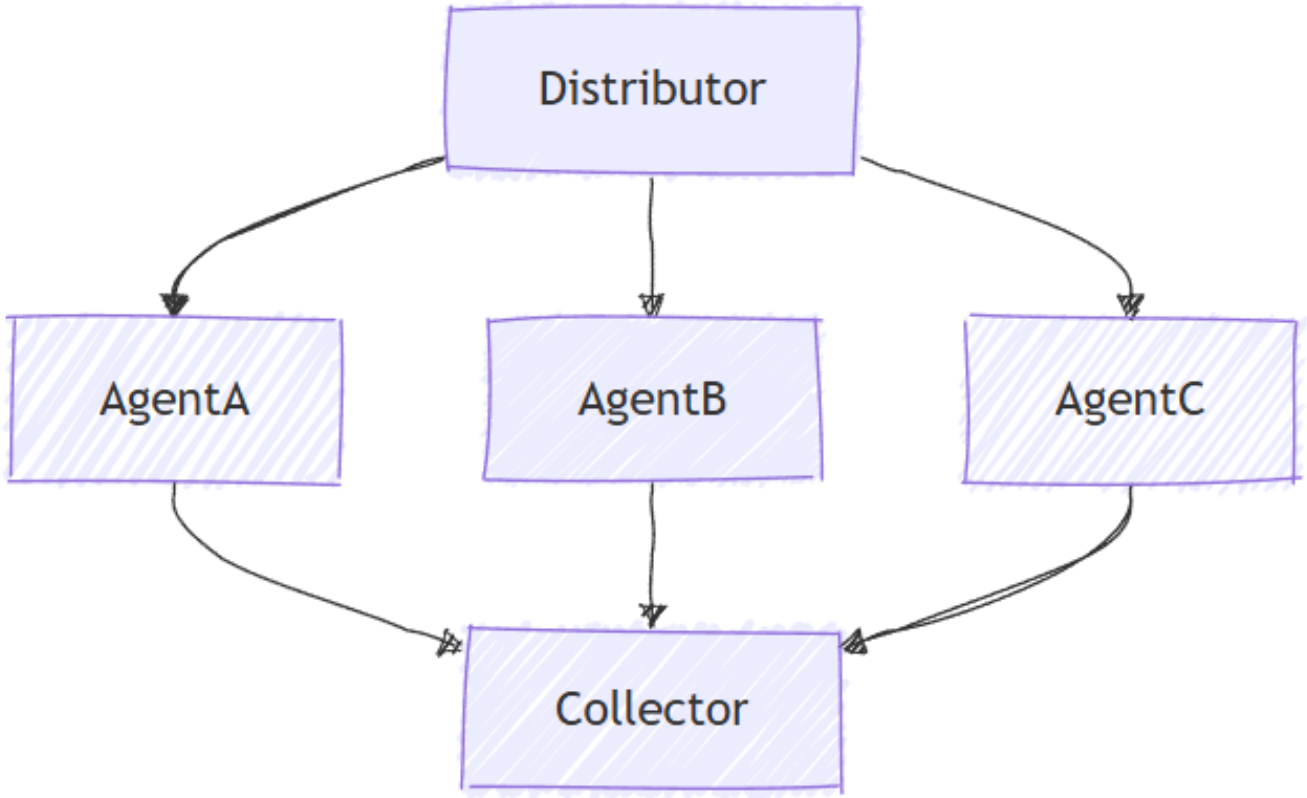
Concurrent

Caratteristiche

- Tutti gli agenti lavorano contemporaneamente sullo stesso input
- I risultati sono aggregati (default: lista di messaggi)
- Puoi fornire un aggregator custom

Quando usarlo

- Analisi da prospettive diverse (legale, marketing, tecnico)
- Ensemble reasoning / voting
- Brainstorming parallelo
- Raccolta dati da fonti indipendenti



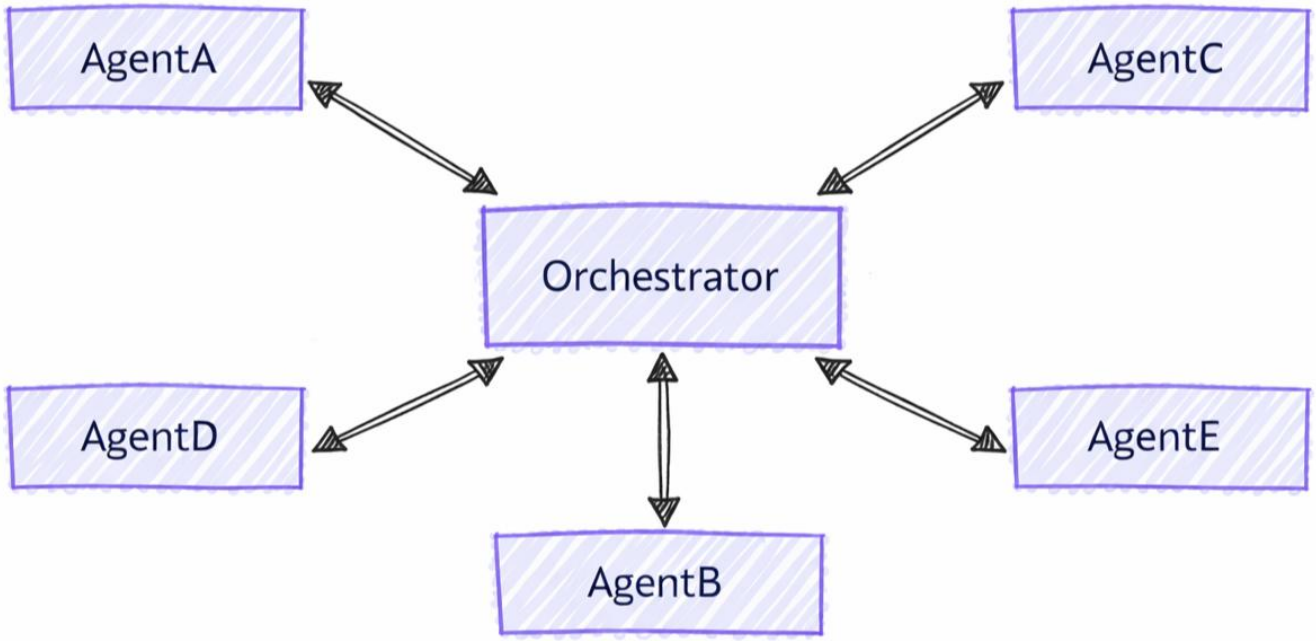
GroupChat

Caratteristiche

- Topologia star: un orchestrator centrale coordinall manager decide chi parla dopo (round-robin, prompt-based, o custom)
- Iterazione: gli agenti possono parlare più volte
- Contesto condiviso: tutti vedono l'intera conversazione

Quando usarlo

- Revisione iterativa (writer ↔ reviewer in loop)
- Problem-solving collaborativo
- Content creation con feedback multipli
- Quality assurance automatizzata



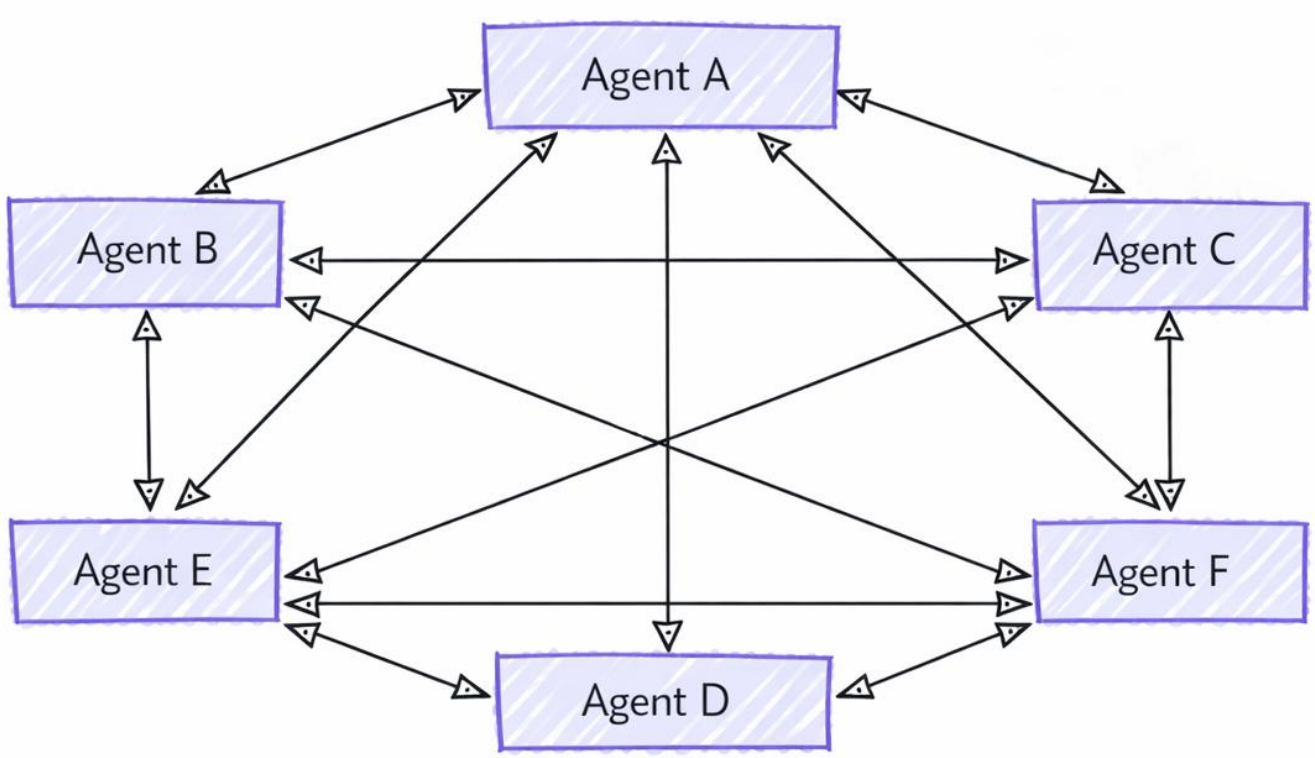
Handoff

Caratteristiche

- Topologia mesh: agenti connessi direttamente, nessun orchestrator centrale
- Ogni agente decide quando e a chi passare il controlloIl ricevente ha ownership completa del task
- Contesto completo passato ad ogni handoff

Quando usarlo

- Customer support multi-specialista
- Escalation e fallback
- Expert routing dinamico
- Workflow dove il percorso dipende dal contenuto della conversazione



Orchestration

Pattern	Topologia	Chi decide il flusso?	Use Case
Sequential	Pipeline lineare	Predefinito (tu)	Task dipendenti, pipeline di processing
Concurrent	Fan-out/Fan-in	Predefinito (parallelo)	Analisi indipendenti, voting, ensemble
Group Chat	Star (hub centrale)	Manager (round-robin o custom)	Brainstorming, revisione iterativa
Magentic	Star con planner	Manager AI con planning dinamico	Problemi complessi e open-ended
Handoff	Mesh (peer-to-peer)	Gli agenti stessi	Customer support, escalation, routing

Guardrails

Gli agenti AI possono chiamare tool esterni, scrivere su database, eseguire azioni

Senza controlli:

- rischio di contenuti dannosi,
- attacchi injection,
- data leakage

Necessità di difesa multilivello (defense in depth)

Guardarails

4 categorie principali:

- Hate & Fairness: contenuti discriminatori
- Violence: contenuti violenti
- Sexual: contenuti sessuali inappropriati
- Self-Harm: contenuti autolesionistici

Livelli: Safe → Low → Medium → High

Ecosistema Microsoft

- Azure AI Content Safety: SDK dedicato per content moderation
- AI Red Teaming Agent: per testare vulnerabilità
- Purview DLP: per prevenire data leakage
- Defender for Cloud: protezione AI a livello infrastruttura

UI

AG-UI

AG-UI (Agent-User Interaction Protocol) è un protocollo aperto, leggero e basato su eventi che standardizza il modo in cui gli agenti AI si connettono alle applicazioni rivolte agli utenti.

È stato creato dal team di CopilotKit per risolvere una sfida fondamentale nello sviluppo di applicazioni agentiche moderne

Il Problema che Risolve

Le applicazioni agentiche rompono il semplice modello richiesta/risposta che ha dominato lo sviluppo frontend-backend nell'era pre-agentica

Gli agenti AI hanno caratteristiche uniche:

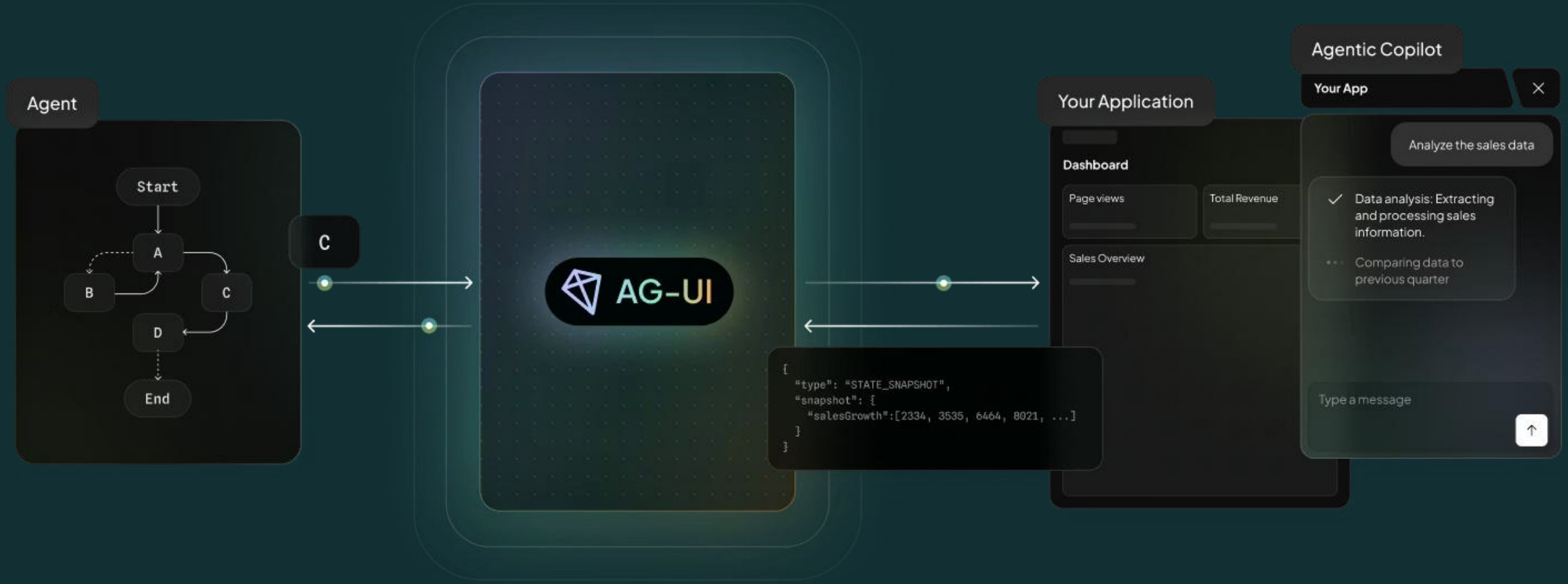
- Sono long-running e trasmettono lavoro intermedio in streaming
- Sono non deterministici e possono controllare l'UI in modo imprevedibile
- Gestiscono simultaneamente I/O strutturati e non strutturati (testo, voce, chiamate a tool, aggiornamenti di stato)
- Richiedono composizione interattiva con l'utente (possono chiamare sub-agenti, spesso ricorsivamente)

Architettura e Funzionamento

AG-UI funziona come un layer di astrazione sopra i protocolli web fondamentali (HTTP, WebSockets), creando un ponte tra le architetture client-server tradizionali e la natura dinamica e stateful degli agenti AI.

Il protocollo trasmette una sequenza ordinata di eventi JSON che include:

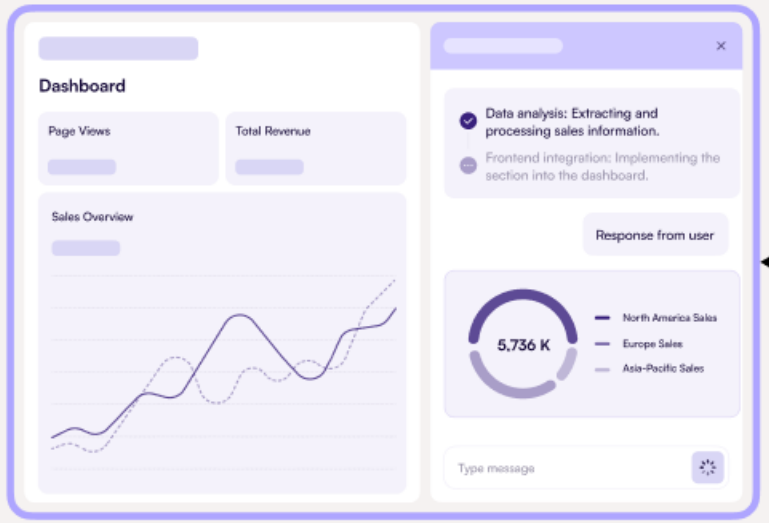
- Messaggi di testo (streaming token per token)
- Chiamate a tool (quando l'agente vuole eseguire azioni)
- Aggiornamenti di stato (state deltas per collaborazione in tempo reale)
- Segnali di lifecycle (avvio, completamento, errori)
- Eventi di interrupt (per approvazioni umane mid-flow)



Supported Agents (1st party integrations & partnerships

mastra Pydantic AI Google ADK LangGraph

Agno AG2 crewai LlamaIndex



AG-UI

Microsoft Agent Framework

Azure AI Foundry

Copilot Studio

CopilotKit

Angular

React

CopilotKit clients

.NET

.NET

Events

UI

State

About & Resources

amengolig/talks/

devpills.net

devromagna.org

Alessandro Mengoli

Technical Leader @ SACMI